AGGRESSIVE AND ENTHUSIASTIC SOFTWARE ENGINEERING

No Longer "Just Writing Code"

LTC Larry G. Baker, USA

The brochure distributed by The United States Army Information Systems Software Development Center - Washington (SDC-W) states that SDC-W has been the leader for the past 30 years in planning, designing, testing, implementing and maintaining the Standard Army Management Information Systems (STAMIS).[1]

However, at times, in many large organizations responsible for the development of computer software, the emphasis mistakenly is placed on the implementing stage of software development. This is an understandable mistake because this is the phase of software development when a tangible product (software code) is produced, and this is the commodity desired by the customer. However, this mistake can be costly in light of the fact that more than 70 percent of the software cost is for software maintenance (correcting the initial software code).

The answer to the problem of "just writing code" is an aggressive and enthusiastic application of software engineering.

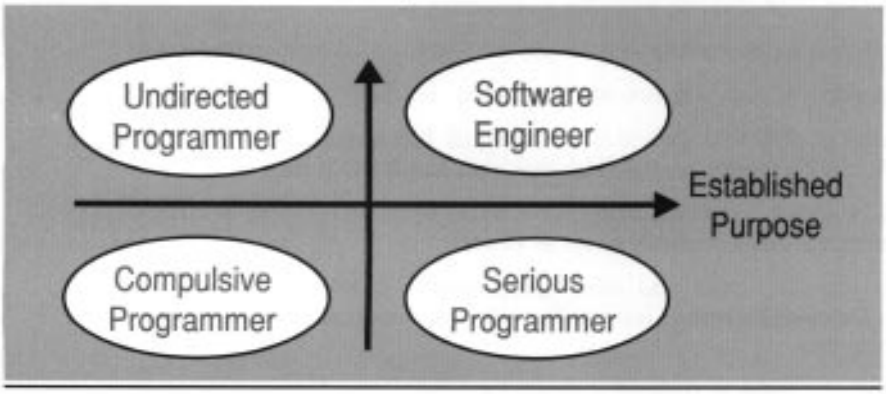
What Is Software Engineering?

A software engineer is one who applies a disciplined engineering approach to software development. The software engineer must be concerned with more than the software solution to the project. Most often, the engineering side of the software engineer is forgotten, and this can be costly in terms of dollars and time. Understanding engineering is more important than understanding coding. That is, software engineers are engineers first and builders of software second; they apply engineering concepts to their work.[2]

The software engineering concept is a fairly new discipline. In the early times of software development, the code was fairly simple, and could be handled easily by one or two people. As the size and complexity of the software projects grew, the people who developed the earlier projects were involved again on these more complex projects because they were successful with the initial programs. This software development style led to the current cliché of the "software crisis." The fundamental cause of the software crisis is that massive, software-intensive systems have become unmanageably complex.[3]

In addition to the complexity of software development efforts, the needs and problems users present are becoming increasingly more perplexing. To ensure the software solutions that are developed are useful to the customer, software developers must

FIGURE 1. Awareness of Real-World Factors



Lieutenant Colonel Baker is Chief, SBIS/ISM Division, U.S. Army Software Development Center-Washington, Information Systems Software Command, Fort Belvoir, Va. He is a PMC 94-1 graduate.

be aware of user requirements and real-world factors. David Marca classified as either compulsive or serious programmers the original software developers who performed software development as programs became more complex.

According to Marca, four different types of programmers exist. Programmers lacking purpose and awareness of real-world factors can be classified as compulsive programmers. Programmers aware of real-world factors but with no goals are undirected in their work. Programmers with a clearly established purpose are serious programmers; they approach the computer intending to obtain useful results. But, serious programmers are not software engineers if they are not aware of real-world factors. Figure 1 summarizes these programming personalities.[4]

The deficiency of not knowing the real-world factors causes the serious programmer to create software that fails to meet customer demands. The principles of software engineering attempt to bridge this difficult gap in building well-engineered software. This ability to identify and assess practical, real-world factors are key to software development. Separating software engineers from other programmers is their ability to make decisions with practical issues in mind during all software development phases.[5]

The software engineer must take user needs and problems and apply various goals (discussed later in this article) and real-world factors to produce a software solution for the user. To produce the correct software solution, the software engineer must use an iterative process with software development cycle to ensure the programmer's final solution is what the user really wants. Only by applying software engineering concepts and goals can the final product be useful to the end user. Figure 2 summarizes inputs and outputs for the software engineer.[6]

The reliability factor for software must be initiated early in its development, which begins with understanding user needs.

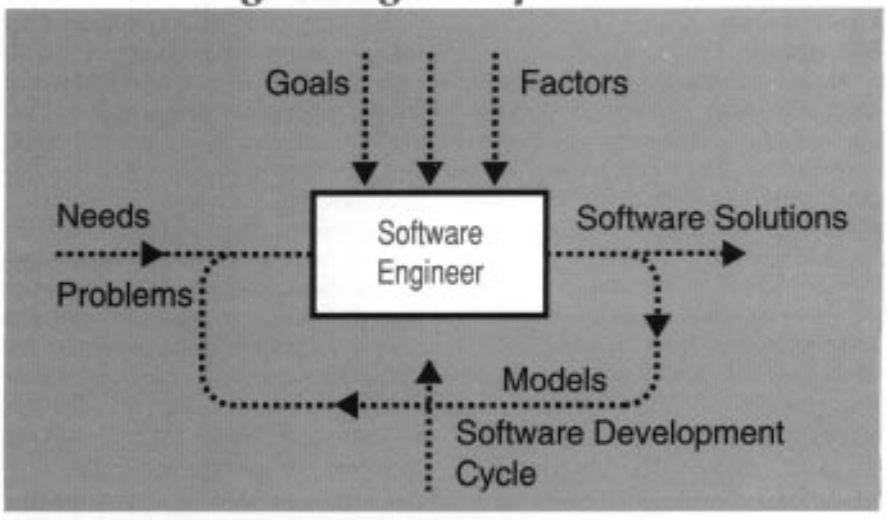
Software Engineering Goals and Purposes

Software engineers must have established purposes and goals when developing software. Purposes and goals are the key ingredients separating programmers from software engineers. In general, the six engineering goals or purposes that must be met to ensure the successful development of a computer software project are: timeliness, efficiency, reliability, simplicity, modifiability and cost-effectiveness.[7]

The software must be done in a timely manner to ensure that the system being developed is not obsolete prior to fielding. Also, if the software development time/cycle as shown in Figure 2 is too long, then user requirements will change to meet the new environment in which they are working. At times, the efficiency requirement is overemphasized. It is an important goal, but only in the context of the system being developed. If the emphasis is to save one nanosecond of CPU time, this efficiency will not be realized by the user, even in a real-time system. Often, this efficiency can be postponed until later in the software development cycle when the entire system is more mature and a better decision can be made regarding software efficiency.

Reliability is a crucial goal for the software engineer. The software must be developed so it will respond correctly to the user's needs and problems. The reliability factor for software must be initiated early in its development, which begins with understanding user needs. A serious problem with reliability occurred in one of the first versions of the Ballistic Missile Early Warning System (BMEWS). The initial version of this critical software detected the rising of the moon as an moving object over the horizon and, according to the software algorithms, the moon was identified as an unknown, hostile target.[8] The software performed exactly as requested; the problem was, as the

FIGURE 2. Engineering Principles



user described, in not identifying correctly the criteria for designating a "target" as hostile or friendly.

The user must be kept in the development loop to ensure the software under development is meeting his needs. Two of the primary goals of software engineering — simplicity and modifiability — are attributed to the user and his ability to use the software. Ease of use is a vital characteristic for any software; therefore, the user interface to the software must be designed carefully. The user is not interested in the internal workings of the software; however, user interface can be one of the most critical factors in determining the overall success of the system.

This view of the overall software package is backwards to the typical software developer who looks primarily at the software from the internal workings outward to the user interface. As the user becomes more adept at using the system, he will begin to want other software enhancements or changes to make his job even easier. The software engineer must be aware of this phenomena and develop software that will be easy to modify in the future. This will help reduce the software maintenance cost, which can be two-to-three times more expensive than the original cost of developing the software.

Government Standards

Help and assistance is available for the software engineer in developing software that meets government standards. The most important government standard for providing guidelines for software engineering is DoD Standard 2167A, "Defense System Software Development," the principal guide for developing software to government standards. This document describes software-specific requirements of system engineering, shows how software fits into the "big picture," and provides detailed descriptions of all documents (Data Item Descriptions) that must be produced by the software engineering team.

The user is not interested in the internal workings of the software; however, user interface can be one of the most critical factors in determining the overall success of the system.

The DoD-STD-2167A also emphasizes activities to be performed during software engineering, with the activities oriented more toward managing the software-development effort throughout the development cycle, as opposed to technical approaches to software engineering.[9]

Summary

No longer can we in DoD develop software in the absence of the stimuli from the user and other real-world factors. Computer programmers cannot isolate themselves from the users, and develop software from an isolated point of view. Only by applying the software-engineering principles can we hope to stop or at least slow the "software crisis." Only by enforcing a disciplined engineering approach on the software development can future computer systems be developed that will meet end-user needs. Only by using software engineering can we eliminate the concept of the "black art or wizardry of software development."

If we are to develop better software (of quality, on time, under budget), we must expand our horizons from the narrow view of software programmers (writing code) to a more expanded view of the software engineer. We must follow a disciplined engineering approach to develop software. This approach is software engineering.

Endnotes

1. COL C.B. Mitchell, "Making Quality Software Happen," Software Development Center - Washington 1993: p. 2.
2. David Marca, *Applying Software Engineering Principles* (Boston: Little, Brown and Company, 1984) p. 3.
3. Grady Booch, *Software Engineering with Ada* (Menlo Park: The Benjamin-Cummings Publishing Company, 1987) p. 28.
4. Marca, p. 6.
5. Marca, p. 4.
6. Marca, p. 2.
7. Booch, p. 29.
8. Marca, p. 12.
9. William H. Reotzheim, *Developing Software to Government Standards* (Englewood Cliffs: Prentice Hall, 1991) p. 161.

Bibliography

Booch, Grady. *Software Engineering with Ada*. The Benjamin-Cummings Publishing Company, Inc., Menlo Park, Calif., 1987.

David Marca. *Applying Software Engineering Principles* . Little, Brown and Company, Boston, Mass., 1984.

Mission Critical Computer Resources Management Guide. Defense Systems Management College, Fort Belvoir, Va., 1992.

Rotzheim, William H. *Developing Software to Government Standards*. Prentice Hall, Englewood Cliffs, N.J., 1991.

Vick, C.R. and C.V. Ramamoorthy. *Handbook of Software Engineering*. Van Nostrand Reinhold Company, New York, N.Y., 1984.